CPF Prototype

Architecture

The CPF prototype is based on the Apache Object Oriented Data Technology (**OODT**) open source data management system framework. In particular the sub project named cas-workflow provides a workflow manager which effectively becomes the CPF Workflow Server. The workflow manager server accepts incoming events and manages execution of workflows and their tasks. It is a stand-alone Java application using XML-RPC (XML messages over HTTP connections) as the server protocol.

The OODT workflow manager uses a resource manager to execute tasks remotely (i.e. outside of the local Java VM). Here is where we diverge from OODT, which uses the cas-resource sub-project to manage compute resources. The cas-resource sub-project:

- Provides an XML-RPC resource manager server to perform job scheduling
- Provides XML-RPC servers at each execution host to execute jobs, and
- Integrates the Ganglia cluster management software to monitor execution hosts

Instead of using cas-resource as the execution back end for remote tasks we instead use the Univa Grid Engine (UGE). UGE is a commercial product that provides features similar to the OODT resource manager.

OODT - UGE Integration

Resource Manager Integration

UGE resource management is integrated by using the class *UGEResourceManagerClient* which replaces the OODT *XmlRpcResourceManagerClient*. Both classes allow the workflow server to submit jobs, delete jobs, query job status, and manage job queues. Instead of communicating with the OODT resource manager server, the UGE client uses the UGE REST API to communicate with the UGE cluster.

We prefer to replace the *client* side of the resource manager because replacing the *server* side means that we would need to develop yet another server to receive client messages and send them to UGE. This adds an unnecessary link to the communication chain between the workflow server and UGE and introduces a new point of failure.

Unfortunately, OODT version 1.0 was not designed to allow projects to easily replace the resource manager *client*. Replacing the client is nonetheless still possible by also replacing the OODT classes that themselves construct the client. This is not ideal so in the future we will submit a git *pull request* to hide the resource manager client behind an interface and instantiate it via a factory class.

UGE Task Integration

To allow OODT workflows to execute arbitrary jobs we created an workflow task called *ExecutableTask*. This task encapsulates (primarily non-Java) scripts or applications that can be executed on the Univa cluster. *ExecutableTasks* can also run in OODT configurations not using a UGE resource manager, via normal Java fork/exec. The task configuration must specify special properties for the command path and its working directory. Other configuration properties are passed via command line arguments. OODT workflow metadata is supplied as environment variables that are available for the command to read.

In the future we may also want to allow ExecutableTasks to modify workflow metadata. This could be done with a support library or a wrapper that looks for changes to environment variables.

OODT Task Integration

Normal OODT workflow tasks are classes written in Java and implement the *WorkflowTaskInstance* interface. UGE cannot invoke these tasks directly so we wrote an OODT workflow task wrapper to provide this feature. This allows new and legacy OODT workflow tasks to run on the Univa cluster.

The current wrapper implementation invokes a Java executable to calls the task "run" method with the correct arguments and metadata.

This will not be very efficient if in the future we need to execute a large number of short-lived Java tasks. In that case we may develop a durable server to run at each cluster node.

Issues

UGE REST API

Adding environment variables to REST-submitted jobs is not working. This will need to be fixed since OODT Metadata will be sent to tasks via environment variables.

Also, the UGE REST interface seems to be incomplete. In particular, it doesn't seem possible to distinguish between...

1. Jobs that have been submitted but are not yet queued or running
2. Jobs that have been submitted but can't run because their command doesn't exist on the system, and
3. Jobs that have been submitted and ran to completion

...because all three conditions return the same error message "Job not found".

My current workaround is to busy-poll the job status right after I submit a job until I get back a valid status. That way, I can be "certain" (is this true???) that the job found its way into a queue for execution.
We started with UGE REST thinking that would be a simple and effective way to communicate with the cluster. If we can't work around these issues satisfactorily we have at least two additional options for communicating with the cluster:

- Use the standard DRMAA/DRMAA2 cluster management API. We discount this option as the API seems very complicated and other teams at NASA have had issues with session stability
- Fork/exec the UGE CLI tools (qsub, qstat, etc.) from inside the workflow server. The CLI tools are able to query historical jobs which the REST API apparently cannot. This option is not ideal for testing since it requires the workflow server to run on the UGE cluster master.
- The UGE tools (CLI tools & UGE REST server) use an internal API to communicate with the cluster. We could possibly use this API in our resource manager also. Of course this carries a large risk of obsolescence when Univa releases new software.

OODT

The current OODT implementation of the "Engine" that runs workflow tasks uses a thread pool but only runs one task at a time from each workflow. So this essentially means that individual workflows can't use parallelism. This limitation is probably OK for now but should be addressed eventually.

Checkout & Build

Follow these instructions to checkout the GIT repository, build the prototype, and install it on your MAC OS X laptop.

GIT Repository

Clone the cpf repository:

```
% git clone https://git.earthdata.nasa.gov/scm/mii/cpf.git
```

Maven Build

Use Maven to build the project from command line. This will build all software and create a "snapshot" release of the controller for your platform. Skip the unit tests for now because they currently require a Linux login credential that is not in the repository:

```
% mvn package -Dmaven.test.skip=true
```

If you are on a mac and you want to build a release for linux:

```
% mvn package -Dmaven.test.skip=true -Plinux
```

Installation

After a successful Maven build you should have a controller distribution:

```
% ls -tla  target/clarreo*dist*

-rw-r--r--  1 abartle  NDC\Domain Users  25796117 Mar 15 13:26
target/clarreo-0.0.1-SNAPSHOT-mac-dist.zip
-rw-r--r--  1 abartle  NDC\Domain Users  25777793 Mar 15 13:26
target/clarreo-0.0.1-SNAPSHOT-mac-dist.tar.gz
```

Untar/unzip the distribution to an installation folder:

```
% cp target/clarreo-0.0.1-SNAPSHOT-mac-dist.tar.gz ~/CLARREO
% cd ~/CLARREO
% tar -xzf clarreo-0.0.1-SNAPSHOT-mac-dist.tar.gz
```

The distribution includes the following folders:

- etc – system configuration
- policy – workflow configuration folders
- bin – commands to start & stop controller and send events
- lib – required java libraries
- logs – log files go here

The OODT/UGE example workflow configuration uses the UGE REST interface to execute jobs for testing. You will need to add UNIX credentials to the system configuration. The credential used here must also be a member of the UGE "sudoers" group on the cluster master cpf.larc.nasa.gov:

```
% vi etc/oodt_examples_uge.properties
_____________________________________________________________

...
org.apache.oodt.cas.workflow.engine.resourcemgr.ugeuser=[UNIX user, member
of UGE sudoers ACL]
org.apache.oodt.cas.workflow.engine.resourcemgr.ugepass=[UNIX password]
...
```

Test

Import cpf.larc.nasa.gov Certificate

The UGE REST server is running on host cpf.larc.nasa.gov and uses a self-signed certificate that your Java interpreter is skeptical of. So you will need to import the certificate to your java keystore.

Follow the instructions How to fix certificate errors with MIIC OPeNDAP servers, using the server https://cpf.larc.nasa.gov:8183 in place of the OPeNDAP server.

Sleep Workflow Test

Start the workflow manager. You may set the environment variable CAS_WORKFLOW_PROPS to the desired workflow .properties file to avoid having to choose it each time you start.

```
% cd bin
% ./wmgr start
[ select workflow configuration if necessary ]
```

In another window, send an event named "sleep" to the workflow manager running on the same host:

```
% ./wmgr-client -op -se -en sleep -url http://localhost:9001
```

The sleep event is defined in policy/examples/events.xml to trigger the sleepWorkflow:

```
<event name="sleep">
  <workflow id="urn:cpf:sleepWorkflow" />
</event>
```

The sleepWorkflow is defined in policy/examples/sleepWorkflow.workflow.xml. The workflow runs a single Sleeper task. Task preconditions and other follow-on tasks (if any existed) would go here.

```
<cas:workflow xmlns:cas="http://oodt.jpl.nasa.gov/1.0/cas"
name="sleepWorkflow" id="urn:cpf:sleepWorkflow">
 <tasks>
   <task id="urn:cpf:Sleeper"/>
 </tasks>
</cas:workflow>
```

The Sleeper task is defined in policy/examples/tasks.xml. It identifies the command to run. Configuration properties will be supplied as command-line arguments. The special configuration properties "Executable" and "WorkingDirectory" are used to identify the command and its working directory:

```
<task id="urn:cpf:Sleeper" name="Sleeper"
class="gov.nasa.clarreo.controller.oodt.ExecutableTask">
  <configuration>
     <property name="duration" value="10"/>
     <property name="Executable" value="/homedir/abartle/bin/sleeper.sh" />
  </configuration>
</task>
```

You can verify this worked by examining the workflow manager output. You should also see output of the Sleeper task in your home directory on

cpf.larc.nasa.gov. The name of the output file should be [*Task URN*].[*Task ID*], for example:

```
-bash-4.1$ more urn_cpf_Sleeper.294
sleeping 10 seconds...
...done!
```

Eclipse Setup

Eclipse users can import project cpf-controller into Eclipse as a Maven project:

- File Import, Existing Maven Project
- Root directory: [repo location]/cpf/cpf-controller

Debugging

Create a configuration to run the workflow manager inside the Eclipse debugger. Although you are running inside eclipse the prototype must first be installed as described in Installation.

Create a Debug target for the workflow manager:

- Run Debug Configurations...
- Java Application (right-click) New
- Main tab
    - Name: Workflow Manager
    - Project: cpf-controller
    - Main class: org.apache.oodt.cas.workflow.system.XmlRpcWorkflowManager
- Arguments tab
    - Program arguments:

      --portNum 9001

    - VM arguments:

      -Dorg.apache.oodt.cas.workflow.properties=${CAS_WORKFLOW_HOME}/etc/oodt_examples_uge.properties
      -Djava.util.logging.config.file=${CAS_WORKFLOW_HOME}/etc/logging.properties
      -Dorg.apache.oodt.cas.pge.task.metkeys.legacyMode="true"
      -Dorg.apache.oodt.cas.pge.task.status.legacyMode="true"

    - Edit Variables...
        - Add variable: CAS_WORKFLOW_HOME [Location of your installation folder]
    - Working directory
        - Set to Other: ${CAS_WORKFLOW_HOME}/bin
- Environment tab
    - Add variable CAS_WORKFLOW_HOME again so the Java program can read it.

Save the target.